UNLIMITED

DEFENCE RESEARCH AGENCY
MALVERN

MEMORANDUM No. 4503 (Issue 2)

LIES, DAMNED LIES AND DATABASES

Author: S Wiseman

DEFENCE RESEARCH AGENCY,
MALVERN,
WORCS.

MEMORANDUM No. 4503 (Issue 2)

UNLIMITED

CONDITIONS OF RELEASE

0127748 311447

*************************** DRIC U

*************************** DRIC Y

Reports quoted are not necessarily available to members of the public or to commercial organisations.

DEFENCE RESEARCH AGENCY, MALVERN

Memorandum 4503

Issue 2

Title: **LIES, DAMNED LIES AND DATABASES**

Author: SIMON WISEMAN

Date: APRIL 1992

ABSTRACT

A database is usually expected to give correct and complete answers to queries. However, some applications take confidentiality to an extreme and require the database to deceive some users by supplying incorrect answers. This paper examines these requirements and studies the effectiveness of three database security techniques in this area.

Lies, Damned Lies and Databases

1. Introduction

A database management system (DBMS) provides services for storing and retrieving information in a way which is logically independent of the physical storage structures employed. In a secure database, where confidentiality is of prime concern, the stored information is ascribed various classifications and there is a requirement that no user, or process running on their behalf, may obtain information unless their clearance dominates its classification.

In addition to ensuring that information is not directly given to users with insufficient clearances, the DBMS, like any other secure system, must ensure that classified information is not leaked indirectly. Highly classified information could be encoded, using the facilities of the DBMS, in a way which makes it appear to be of lower classification. Users with low clearances who know the encoding scheme would then receive highly classified information from a Trojan Horse operating at the higher level.

An additional requirement is that the database schema must be inferentially secure [Morgenstern88], that is highly classified information can never be inferred from lowly classified information. Solving this inference problem, and the allied aggregation problem, is the concern of the database design process, and has been dealt with elsewhere [Lunt89].

So it is necessary for both the design of the database schema and the DBMS to be secure. However, confidentiality is not the only security problem. Integrity is an important concern and is partly addressed through the use of constraints in the database schema. The use of constraints can be likened to the use of defensive programming techniques. They are the first line of defence for integrity as they ensure the database is always in a valid state, though they do not guarantee that this state is appropriate [Terry89]. By offering constraint enforcement as a general service, a DBMS makes the implementation of robust applications more cost effective.

One important reason for using a DBMS to store information is, therefore, that it helps preserve the integrity of the information it holds. However, there are applications where different users are required to have views of the same information which, while they are individually self-consistent, actually contradict each other. In particular, a database can be required to lie about the true state of the world to some users. At first sight this appears to be a requirement for databases which have no integrity, but actually the database must lie consistently and properly, and the integrity of the lies is extremely important.

Most of the examples of this kind arise in the military intelligence arena, however a good example occurs in the medical world[1]. On reaching a conclusion about a patient's condition, a Doctor might tell the patient they have Bronchitis and send them off for hospital tests. However, the Doctor tells the hospital that lung cancer is suspected. The deception must be maintained to avoid premature concern in the patient and embarrassment to the Doctor.

This paper examines some of the requirements for databases which are intended to deceive some users and compares the effectiveness of three secure database implementation techniques in this kind of application. Section two introduces some example requirements for deceptive databases while section three informally presents security models for three implementation techniques. In section four, the suitability of the three techniques is examined with respect to applications which deceive with integrity. Finally, section five summarises the position.

[1] This example was given by John Dobson at the 1990 IFIP WG11.3 Database Security Workshop to illustrate a point from [Martin90].

2. Deception

There are requirements for various different kinds of deception in a secure database. These range from "white lies", which seek to protect the innocent from the sordid truth, to the really deceitful lies which aim to mislead and corrupt. This section discusses the various possibilities and illustrates them with examples, but first it considers what the responses of an honest database should be.

2.1 Honest Replies

Typically, a database is interrogated by requests that ask for all information meeting certain criteria, such as "how far is the Space Ship Comet from Earth?". To such questions an honest secure database would reply either with the required information, or with notification that the user has insufficient clearance to calculate the result, or with notification that the resulting information itself is too highly classified, or perhaps some combination of all three.

The following example is given to illustrate these three kinds of honest response. The notation of the fact model is used [Sowerbutts90], and the facts, question and replies are summarised in Figure 2.1.

The facts.

Unclassified:	Comet is going somewhere
Confidential:	Comet is going to Earth
Secret:	Comet is 42.59 parsecs from Earth

The question.

"How far is the SS Comet from Earth?"

The answers.

Unclassified:	you have insufficient clearance to calculate answer
Confidential:	you are not cleared to know the distance
Secret:	42.59

Figure 2.1: Example facts and honest answers to a question.

So the Space Ship Comet is going to Earth and is currently at a distance of 42.59 parsecs, and this information is Secret. However, the fact that the Comet is going somewhere is Unclassified and that its destination is Earth is Confidential. Users who ask "how far is SS Comet from Earth?" will get different answers depending on their clearance.

A user with a clearance of Unclassified would get the answer "you have insufficient clearances to calculate the answer". This is because the DBMS must check that Comet is actually going to Earth, but it finds that a Confidential fact must be examined to ascertain whether this is true. Therefore the reply cannot be "Comet is not going to Earth" because it might be, so the DBMS gives the non-committal, but truthful, reply which essentially says it doesn't know (at that level of clearance). Note that the same answer would be received if the Unclassified user asked "how far is SS Comet from Vesuvius?".

A Confidential user would receive "you are not cleared to know the distance". This is on the basis that the DBMS can tell that Comet is going to Earth, but when it goes to find out the distance it discovers the user is not cleared to see it. Thus the DBMS is able to give a reply which confirms that the Comet is heading for Earth, but the actual distance cannot be revealed. Again this is an honest reply, and is more detailed than that given to the Unclassified user, but it is not the complete answer.

A Secret user obviously gets the answer "42.59" which is the whole truth. This does imply the fact that Comet is going to Earth, but this fact is only Confidential and therefore no leak has occurred.

2.2 Denying Existence

Research has shown that Proton Missiles, carried by all Space Ships in the Community's fleet, may be hazardous to the health of the crew. However, to avoid lowering morale it is necessary not only to keep the details secret but also to classify the very existence of the information. Anyone without the appropriate clearances who asks about the health hazards of Proton Missiles will be told "there is no information of that kind".

An example of this is shown in Figure 2.2. The database deceives Unclassified users, by claiming that no such information is stored. Confidential users get an honest answer, to the effect that there is no information of that kind available to them but there is information which they cannot see. Secret users are told that the information does exist, but they are not allowed to know the details. Only Top Secret users get the complete truth.

The facts.

Unclassified:	nothing is known
Confidential:	there is some Secret information
Secret:	there is data about radiation from the Mk4 missile
Top Secret:	Mk4 missile emits 94.8 femtoUrgs per second

The question.

"How much radiation does the Mk4 missile emit?"

The answers.

Unclassified:	there is nothing about radiation from MK4 missiles
Confidential:	if that information exists, you cannot see it
Secret:	you are not cleared to know the radiation output
Top Secret:	94.8

Figure 2.2: Example of denying the existence of information.

Note that the reply to Unclassified users categorically denies the existence of the information, which is a complete lie, whereas Confidential and Secret users receive truthful, but incomplete, answers.

2.3 Cover Stories

The invasion of neighbouring Romantic space by the Community needs to be a well kept secret, since success relies on complete surprise. So when SS Magna Carta is loaded with invasion troops and sent to Romance, a cover story has to be invented to avoid suspicion.

The facts.

Unclassified:	Magna Carta is going to Romance	
Confidential:	Magna Carta has a covert mission	
Unclassified:	Magna Carta is going to Romance to	(deliver aid
Secret:		invade)

The question.

"Why is Magna Carta going to Romance?"

The answers.

Unclassified:	to deliver aid
Confidential:	low users think its to deliver aid, but its really something covert
Secret:	low users think its to deliver aid, but really its to invade

Figure 2.3: Example of a cover story.

Thus the general public are told that Magna Carta is on a mercy mission to deliver aid, while highly cleared users are told of its real purpose. It is possible that there are some users who know the mission is covert and that a cover story has been created, but they cannot find out details of the real mission. Figure 2.3 shows the example, with the cover story shown by enclosing the different versions in brackets. This is to emphasise the difference between having two contradictory facts and having a cover story.

In this example, a Space Ship can be heading to only one destination at any one time and this can only be for one reason. If the cover story was simply given as an additional fact at the Secret level, as in Figure 2.4, this would contradict the Unclassified fact. This contradiction violates the integrity constraint of unique destination and purpose per ship.

Thus in figure 2.3 the notation shows that the Magna Carta is going to Romance to invade, and this is Secret. It also says that a cover story exists, which can be seen by Unclassified users. The story is that the Magna Carta is going to Romance to deliver aid. By contrast, figure 2.4 shows the Magna Carta is going to Romance to invade and deliver aid. There is no information available to indicate that the Unclassified fact represents a cover story, since it may simply be that inconsistent data has been entered.

Unclassified: Magna Carta is going to Romance to deliver aid
Secret: Magna Carta is going to Romance to invade

Figure 2.4: Facts which violate integrity.

So the integrity constraint which insists each ship has a unique destination and purpose is important. Without it an errant user or application software would be able to send Magna Carta to Romance for *refueling* at the same time as it is going to *invade* (or rather *deliver aid* if you are not cleared to Secret). The use of weakened forms of integrity, such as Polyinstantiation Integrity [Denning88], go some way to avoiding this problem, but these do not stop an errant program at a different security level entering the contradiction.

2.4 Secrecy about Changes

Eventually, when the Magna Carta reaches Romance, the cover story is revealed, but later the SS Adventurer heads for Romance to support the ill fated invasion. This is common knowledge, however, on route the Adventurer falls prey to a previously unencountered life form. Back at HQ, intelligence sources realise that Adventurer isn't going to make it to the battle. However, this information must be kept Secret since the Romantics are currently falling back and the Community troops are pressing forward, all because they think the Adventurer is coming.

The facts.
Unclassified: Adventurer is going to Romance
Unclassified: Adventurer will reach Romance in (1 hour
Secret: 1 week)

The question.
"When will Adventurer reach Romance?"

The answers.
Unclassified: 1 hour
Secret: low users think its 1 hour, but really its 1 week

Figure 2.5: Example of secrecy about changes.

This example is just like a cover story, though it arises in a different way, and is shown in Figure 2.5. As far as the Unclassified Community troops and any eavesdropping Romantics are concerned, the Adventurer is about to arrive. However, the commanders know that it has been seriously delayed and can make suitable plans.

The difference between this kind of example and cover stories is that here the Unclassified fact started off as the truth. Only later did the truth change to a highly classified fact and the low information became a cover story. With a cover story, the truth is initially highly classified and a misleading version is deliberately entered with a lower classification.

The facts.

Unclassified:	Adventurer is going to Romance	
Secret (Tactical):	Adventurer will reach Romance in	1.084 weeks
Secret (Strategic):		1 week

The question.

"When will Adventurer reach Romance?"

The answers.

Unclassified:	you are not cleared to know
Secret (Tactical):	1.084 weeks
Secret (Strategic):	1 week

Figure 2.6: Example of a précis.

The example shown in Figure 2.5 gives different answers depending on the user's clearance, though it is also possible that the answer could depend on some other attribute of the user, such as their role. This is useful if the requirement is to hide changes which occur frequently, as it can be used to present certain users with a précis of some information. This kind of "white lie" is quite different to a cover story, since it can actually be beneficial because it hides rapidly changing irrelevant detail.

Figure 2.6 gives an example of a précis which is based on whether the user is playing a Tactical or Strategic role. The tactical information is accurate but will be changing rapidly, though most changes will be small and irrelevant to the Strategic user who wants a more global picture. Obviously some integrity constraints would be required to ensure that the Strategic picture does not become too out of line with the Tactical information, but this concern is not addressed here.

In this example, the Secret users ask the same question but receive a reply which depends on their role. Although they need not know of the other's existence, it is not of paramount importance to keep the other hidden. This differs from cover stories where hiding the existence of a cover can be vital.

A problem that occurs with rapidly changing information concerns the ability to serialise concurrent transactions. From an integrity point of view it is essential that all concurrent execution of transactions can be seen as some serial execution of the transactions. Generally, schemes which ensure this are the basis for covert channels in a secure DBMS, but proposals have been made for secure serialisation [Keefe90]. These however are prone to availability problems, whereby a highly cleared user wishing to obtain a consistent picture of lowly classified information may repeatedly be rolled-back because the low information keeps changing. The use of a précis would reduce the frequency of changes and make it more likely that the high users could complete their work.

3. Secure Database Models

In this section, three techniques for database security are described. Rather than describe them in terms of a data model and argue that they are secure, a security model is used to model them and this is related to the relational data model as a justification of the model's appropriateness. The use of a model to describe the way in which the techniques work is necessary to enable the confidentiality controls, on which the integrity of the deceptions rely, to be clearly identified. Also it avoids describing specific secure DBMSs, which necessarily introduce additional constraints and features which are not directly relevant to this discussion.

A simple Bell-LaPadula style of security model [Bell74] is used. The important aspects of this model are that:

1). Objects are classified containers and a policy of "no flows down" is enforced;

2). No flows down is not violated by transitions which involve a "pure write" or "append" kind of alteration of high Objects while also observing or modifying low Objects;

3). Objects may be created and destroyed and an Object's classification may be raised at any time, though an addressing Hierarchy is employed to ensure that no covert channels arise through these operations;

4). An Object's classification may be lowered at any time, subject to the controls of the Hierarchy and as long as its original contents are completely destroyed and the new classification dominates the sources of the new contents;

5). The roots of the Hierarchy are either fixed or their creation and destruction is subject to some unspecified control which avoids the potential covert channels.

The models presented here are abstract interpretations and do not necessarily reflect how such databases are implemented in practice. Also, it is likely that more constraints will be imposed on the database by implementation considerations.

3.1 Polyinstantiation

The most widely proposed and used technique for providing secure DBMSs is called Polyinstantiation[1] [Denning87]. Other flavours of Polyinstantiation have been proposed [Haigh90] [Jajodia90], but all are covered by the model given here.

There are two kinds of Object in this model: collections of facts and schema details. A collection of facts has a classification, which applies to all facts in the collection. The Collection Objects can only be accessed through the Schema Object that describes the facts that they contain. The classification of a Schema Object is always dominated by the classification of any Collection that it refers to.

The Schema Objects correspond to tables in the relational model. The Collection Objects contain single level subsets of the table. If data is classified at the row level, all rows of the same classification are stored in one Collection Object. If fields are separately labelled, data from different Collections must be joined in some way, [Denning87] [Jajodia90], to reconstruct the multi-level information.

A new fact can be added to the database by adding it to one of the collections. This alters one of the Collection Objects and so the user must have a clearance which is dominated by the classification of the collection. The user's clearance must also dominate the classification of the Schema Object in order that they can "address" the Collection Object.

Although this model allows a user to overclassify a fact, by inserting it into a Collection Object whose clearance strictly dominates their clearance, it is unlikely to be implemented because of

[1] Polyinstantiation refers to the simultaneous existence of multiple objects with the same name that are distinguished by their classification [Denning87]. Allowing polyinstantiation is one technique for closing covert channels that arise when creating and deleting objects. Polyinstantiation is not an inevitable consequence of multi-level security.

the difficulty in providing a "pure" write which alters just part of an Object. Thus in practice it is likely that a user will only be able to insert a fact at their own level.

Similarly, a user can delete a fact which is in a Collection whose classification dominates their clearance, but again this is likely to be limited in an implementation to deleting at their level.

Users can observe facts which are in a Collection Object only if its classification is dominated by their clearance.

A high user cannot update a low fact, but Polyinstantiating databases generally change such requests into inserts at the higher level. The low Collection is observed, the fact is modified according to the requested update and then inserted into a collection whose classification is that of the user's clearance. No low Collection Object has been modified so the operation preserves confidentiality.

For example, suppose the database contained the Unclassified fact that "Adventurer is carrying Bananas to Earth". If a Secret user were to change the cargo to Missiles, this would be translated into an insert of the Secret fact "Adventurer is carrying Missiles to Earth".

However, an insert is not possible if an appropriate collection does not exist to hold the new fact. A new Collection Object can only be created by a user whose clearance equals the classification of the Schema Object. This is because the address of the new Collection Object must be written into the Schema Object, which is its parent in the Hierarchy. This problem can be overcome by having a Schema Object per security level, but in this paper the simpler model will be used as it does not materially affect how polyinstantiating databases can be used for deception.

A user who wishes to establish the truth of a fact must be able to observe the appropriate Schema Object in order to determine which Collection Objects need to be examined. If the fact is found then it is true, but if the fact is not found it is either false or too highly classified for the user to see.

In general, some of the facts described by a schema will be too highly classified for the user to see. Thus it is not possible to enforce integrity constraints which can only be established by examination of all the facts. Polyinstantiating databases can therefore only enforce weakened forms of Entity Integrity and Referential Integrity [Burns90].

As an example, suppose that the database contains two Schemas, both Unclassified. The first concerns captains of space ships and says that facts of the form "_ is the Captain of the _" are stored. The second Schema is about space ships' destinations, with the facts having the form "_ is going to _". The actual facts about captains can be either Unclassified or Secret, because the Schema refers to two Collection Objects with those classifications. Similarly for destinations. However, there are currently no Secret facts about captains. This is shown in Figure 3.1. Each box is an Object and the arrows and indentation reflect the Hierarchy.

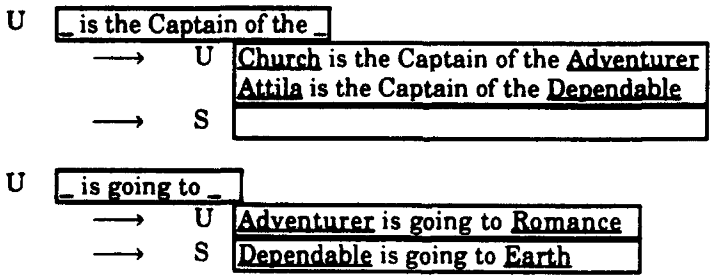
U
_ is the Captain of the _
U
Church is the Captain of the Adventurer
Attila is the Captain of the Dependable
S
U
_ is going to _
U
Adventurer is going to Romance
S
Dependable is going to Earth

Figure 3.1: Two Schemas each with two Collections.

Now suppose that a Secret user wishes to change the destination of the Adventurer to Vesuvius. The user can observe the fact that Adventurer is currently heading for Romance, because the Collection is Unclassified, but is prevented from altering it because this would be a write down. However, the polyinstantiating database treats the user's update as a request to insert a new fact at the higher level. Thus a new Secret fact is placed in the Secret Collection. Figure 3.2 shows the position after this update.

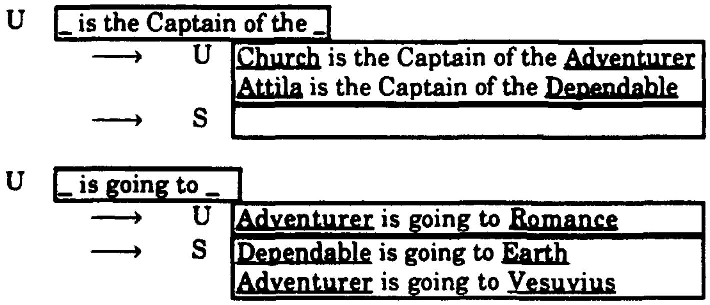
U
_ is the Captain of the _
U
Church is the Captain of the Adventurer
Attila is the Captain of the Dependable
S
U
_ is going to _
U
Adventurer is going to Romance
S
Dependable is going to Earth
Adventurer is going to Vesuvius

Figure 3.2: After updating Adventurer's destination.

Note that the original fact, that Adventurer is going to Romance, still exists. It is visible to Unclassified users, who are unaware that any change has been made. Secret users can observe both the original fact and the new one.

It was the intention that ships have unique destinations, thus this example database has lost its integrity. If the user had logged in at Unclassified and deleted the fact that "Adventurer is going to Romance" and then logged in at Secret and inserted "Adventurer is going to Vesuvius", integrity would have been preserved. However, a low user could always insert contradictory information without knowing it.

Thus with Polyinstantiation, integrity cannot be enforced except by careful design of the application [Burns90]. However, this detracts from the benefit of using a DBMS.

3.2 View Based Classification

A lesser known alternative to Polyinstantiation for enforcing confidentiality in databases is the use of View Based Classifications [Wilson88] [Knode88]. In this model there are Schema Objects and Value Objects. A Schema Object describes a class of facts, such as _is going to_, and gives all possible facts, whether they are true or not. With each possible fact is kept the address of the Value Object which records whether the fact actually is true or not.

In relational terms, a view is a description of all possible tuples that could occur in the view, along with a note of which tuples are currently in the view. A tuple may only be inserted into a view if it is one of the possible tuples. When views are used to classify information, a classification is attached to each possible tuple in the view and this is the classification of the tuple if it becomes part of the view.

In the model, the Schema Object contains a description of a view, by enumerating all the possible facts. However, in practice the view will be described algorithmically and the true/false values will be represented by storing the facts corresponding to the true values. In order to reduce the amount of trusted code required, systems using this technique only allow simple view definitions [Garvey88], but the model describes the general situation.

When a new class of facts is introduced, all the possible facts are calculated and a description is placed in a new Schema Object. For each possible fact, a new Value Object is created and is associated with the possible fact. The creation of all these Objects must be controlled, perhaps using the Hierarchy, in order to avoid a covert channel.

The Value Objects are given a classification which is appropriate for the fact if it were true. This may be higher than the clearance of the user who is creating it. The initial value of the fact is set as appropriate, though obviously this cannot be with a value of *true* if the classification is higher than the clearance of the user, unless such deliberate overclassification is required.

The classification of a fact may depend on information other than its own value, such as the truth of other facts. Thus it is possible that the Schema may refer to several Value Objects for the same possible fact. Integrity constraints will usually ensure that only one of these has the value *true*.

A new fact is added to the database by changing the appropriate Value to true. In order to do this the user's clearance must be dominated by the classification of the Value Object, to avoid a flow down, though usually the user's clearance and the classification will be equal. Note that new Objects are created only when new kinds of fact are introduced, rather than when possible facts are made true. Thus no covert channel arises through inserting new facts. Deleting facts is similar.

Updating involves changing one Value Object to *false* and another to *true*. Since the user's clearance must be dominated by the classifications of both Value Objects in order for them to be altered, it is generally the case that a user may only update a fact "at their clearance". However, as noted by [Garvey88], it is possible for the database to "polyinstantiate" by converting an update of a low fact into an insert of a high fact.

A user who wishes to establish the truth of a fact searches the appropriate Schema Object for a description of the fact of interest. The corresponding Value Object is then examined to find the answer. If the user can observe the corresponding Value Object, then the answer can be determined as true or false. However, if the user's clearance does not dominate the classification of the Value Object the answer cannot be determined.

Generally, integrity constraints can be applied to ensure that those facts which are flagged as true form a consistent picture. Users may only modify the database if they, or the DBMS on their behalf, can determine that the integrity constraints are still met. The proposals in [Wilson88] and [Knode88] ensure this is possible in certain important cases by restricting each fact to having at most one classification.

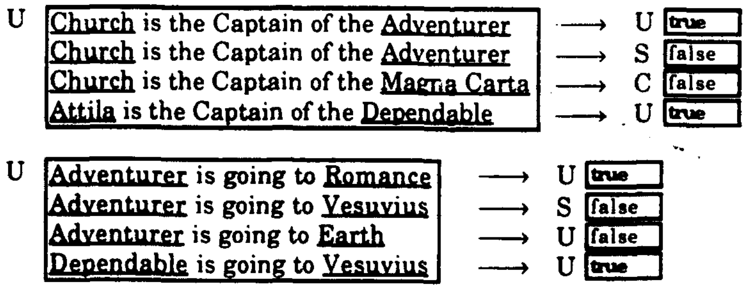
U
Church is the Captain of the Adventurer → U true
Church is the Captain of the Adventurer → S false
Church is the Captain of the Magna Carta → C false
Attila is the Captain of the Dependable → U true
U
Adventurer is going to Romance → U true
Adventurer is going to Vesuvius → S false
Adventurer is going to Earth → U false
Dependable is going to Vesuvius → U true

Figure 3.3: Two classes of fact each with four possibilities.

Now consider the example, shown in Figure 3.3, where the database contains two Unclassified facts. There are actually two classes of fact, described by two Schema Objects. Each describes four possible facts and so addresses four Value Objects of varying classification. Note that the example views do not admit all possible combinations of elements, so for some unspecified reason it is not possible for Attila to be the Captain of the Adventurer.

Assume that Entity Integrity is in force for both schemas, that is a Ship may only have one destination and that a Captain can only be in charge of one Ship. Now consider the example where a Secret user attempts to change the destination of the Adventurer from Romance to

Vesuvius. The user is unable to change the Value Object of "Adventurer is going to Romance" to false, because this is an Unclassified Object and so the operation would constitute a flow down. However, unless this fact is marked as false, it is not possible to change the destination to Vesuvius because of Entity Integrity.

One way out of this impasse is to use polyinstantiation. This allows the Secret user to add a new destination without altering the original. However, this does not preserve Entity Integrity, since the ship ends up with two destinations.

The only method for changing from an Unclassified destination to a Secret one is for an Unclassified user to change the Value Object of the original destination to false and a Secret user to add the new destination. Obviously if this requires a real user to log in at Unclassified and then to change to Secret this will be inconvenient, but the use of a multi-level secure workstation, such as [Cummings87], would undoubtedly improve the user interface.

However, the need to "log in" twice raises a more serious integrity problem in some cases. Suppose the database has an integrity constraint which insists that a ship always has exactly one destination. The problem is that, because two transactions are required to make the update an illegal state, in which the ship has either no destination or two destinations, becomes visible to others.

A potential solution to these problems is to use high water marks, or floating labels [Woodward87], to allow the user's clearance to rise during a transaction. At the start of the transaction the user's clearance is set to Unclassified. The user sets "Adventurer is going to Romance" to false. The classification is now raised to Secret and the user decides on the new destination, Vesuvius, and "Adventurer is going to Vesuvius" is set to true. Once the update has completed, the changes can be committed. This means that no other user sees the database in an inconsistent state, that is the Adventurer always has exactly one destination.

However, this solution is not secure because of the nature of transactions. A user can propose to modify low data and then decide whether to commit or roll back the transaction on the basis of high data. This causes a downward flow because the changes to low data depend on high data. Thus high water marks within a transaction are insecure.

A more subtle problem is illustrated by this example. If the constraint that a ship always has one destination is enforced, once the update has been completed an Unclassified user can infer that Adventurer is heading to Vesuvius. This is because the Unclassified user is able to see that it is heading nowhere else. While this problem can be avoided by suitable data design [Lunt89], it does mean that any schemas designed for such a secure database will have to be evaluated in some way to determine that they do not admit such an inference problem. The need for such scrutiny was recognised by [Wilson88], but such inferences are actually a potential problem in all secure systems, though they become more evident in databases because the data has more structure and classifications are applied with a finer grain of protection.

3.3 Insert Low Approach

Another alternative to Polyinstantiation is the Insert Low approach [Wiseman90a]. In this model there is an Object for each class of fact, which contains schema information, and one for each true fact. The Schema Objects contain the addresses of the Fact Objects, which are their subordinates in the Hierarchy. The Fact Objects referred to by a Schema Object may have various classifications, all of which dominate the classification of the Schema Object. When a new Schema Object is created, the Hierarchy, or some unspecified control, is used to ensure that no covert channel is admitted.

In relational terms, the Schema Object corresponds to a table and the Fact Objects to a row or field, depending on the granularity of labelling. The Hierarchy ensures that users can only detect the existence of fields and rows if their clearance allows them to "pass through" the table. If fields are individually classified, a row is made up of several Fact Objects, each describing the relationship between one subset of the columns. Thus there are generally more Facts per row than there are columns.

When a new fact is inserted a new Fact Object is created, as a subordinate to the Schema Object, and the value of the fact is placed in it. In addition, the address of the new Fact Object must be placed in the Schema Object, thus the user's clearance must equal the classification of the Schema Object. However, the new fact may have a greater classification, though the user would then be unable to see what they have inserted.

So only low users can insert, hence the name of the approach. While this may sound excessively restrictive it actually places no more constraints on the users other than are required for the enforcement of both confidentiality and integrity.

Similarly, to delete a fact it is necessary to remove its address from the Schema Object. This means that only users whose clearance equals the classification of the Schema Object can delete facts.

To update a fact, a user must first find the appropriate Fact Object and modify it. Thus their clearance must be dominated by the classification of the Fact Object. However, the user will normally observe something of the Fact Object's original value, since pure writes are difficult to achieve in practice, and so the clearance will usually equal the classification.

A user who wishes to establish the truth of a fact must examine all the Fact Objects referred to by the appropriate Schema Object. If all the necessary Fact Objects can be observed, the user obtains a complete answer. However, if some have classifications which are not dominated by the user's clearance, the answer may not be complete.

Integrity constraints can be applied generally, but users may only modify the database if they (or rather the DBMS on their behalf) can establish that the constraints are not violated.

For example, in order to determine that Entity Integrity is preserved when inserting a new fact, it is necessary to be able to examine all existing facts to check that the "key" is not already present. The user's clearance must equal the classification of the Schema Object to be able to insert and so if any Fact Object has a classification higher than that of the Schema Object the user will be unable to confirm that Entity Integrity is upheld[1]. Note that a user with higher clearances could check for Entity Integrity but is prevented from inserting because of the "existence" covert channel. That is, unless the "key" facts all have classifications equal to the Schema Object's classification, new facts cannot be inserted.

So although it is not strictly necessary for keys to be single level, on the whole they will be to allow new facts to be inserted. This introduces a potential availability problem, but it is easily controlled by applying integrity constraints to the classifications of the Facts.

Now consider the example of a user wishing to change the Adventurer's destination from Romance to Vesuvius. The fact that the destination is to change is unclassified, while the fact that the destination is to be Vesuvius is Secret. Obviously the user is cleared to at least Secret. Initially the database contains two Schemas, each referring to one Fact. This is shown in Figure 3.4.

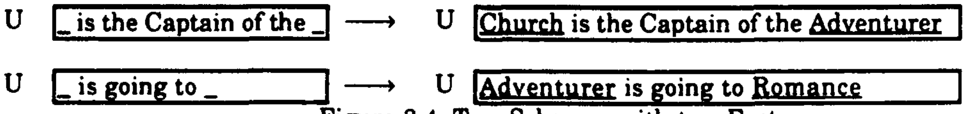
U
_ is the Captain of the _
U
Church is the Captain of the Adventurer
U
_ is going to _
U
Adventurer is going to Romance

Figure 3.4: Two Schemas with two Facts.

[1] Rejecting an insert for this reason does not lead to a covert channel because it is only the value of the fact that is highly classified, not its existence.

So the Secret user needs to change the database as seen by Unclassified users. To avoid downward flows the user must log in at Unclassified and delete the fact that the Adventurer is going to Romance. Next a new Fact Object is created which is classified Secret, even though the user's current clearance is Unclassified. Into this new Fact Object is placed some Unclassified information, namely that Adventurer is going somewhere (effectively a null). The user now logs out and back in at Secret and updates the fact to record the true destination, Vesuvius.

However, suppose there is an integrity constraint which insists that ships always have a destination. The solution just described allows the Adventurer to have a null destination for a short time, contrary to the integrity constraint. An alternative approach avoids this problem. The user logs in at Unclassified and upgrades the Fact Object which records the destination of the Adventurer. Note that changing the classification of a Fact Object is secure if the user's clearance equals the Schema Object's classification, because the Hierarchy hides the change from users with lower clearances. Figure 3.5 shows the database after this action has been committed.

U	_ is the Captain of the _	⟶	U	Church is the Captain of the Adventurer
U	_ is going to _	⟶	S	Adventurer is going to Romance

Figure 3.5: After upgrading the Fact.

Once the Fact Object is classified Secret, the user can log in at Secret and update its contents to the desired value. The final result is shown in Figure 3.6.

U	_ is the Captain of the _	⟶	U	Church is the Captain of the Adventurer
U	_ is going to _	⟶	S	Adventurer is going to Vesuvius

Figure 3.6: After updating the Fact.

With the Insert Low approach the opposite is also possible. An Unclassified user may change a Secret destination to an Unclassified one, assuming that this is reasonable from an integrity point of view, as follows.

The Schema objects will, in practice, contain information about how facts like "Adventurer is going somewhere" and "Adventurer is going to Vesuvius" are related. This would allow an Unclassified user to identify which Fact Object contains the Adventurer's destination, though without revealing the contents of the Fact Object. The Unclassified user may change the classification of the Fact Object to Unclassified as long as they completely overwrite the original information[1]. There is no covert channel because the Hierarchy stops any lower user from seeing the changes.

Figure 3.7 shows the state after an Unclassified user has changed the destination of the Adventurer to Earth. Note that this is secure since the Unclassified user learns nothing of its previous destination and that the Hierarchy ensures no user with lower clearances sees the classification change. Note that this form of "pure write" can be implemented since a complete Object is overwritten.

U	_ is the Captain of the _	⟶	U	Church is the Captain of the Adventurer
U	_ is going to _	⟶	U	Adventurer is going to Earth

Figure 3.7: After updating the Fact to a lower value.

[1] A low user overwriting high data is not a confidentiality problem ([Bell74] allows it!). Like any alteration of the database, even high users overwriting high data, it is likely that it would be governed by integrity controls, such as described in [Wiseman90b].

These methods of updating not only avoid polyinstantiation but preserve integrity constraints which insist that facts of a certain form always exist. The cost of avoiding polyinstantiation is that the user must log in and out, though this would be alleviated by utilising a multi-level workstation, but the benefit is that the database's integrity is preserved and that this can be enforced by the DBMS.

4. Deception with Integrity

In this section the three techniques for database security are examined with regard their suitability for applications that require some users to be deceived. The important issue is how well they enforce the integrity of the deception. If integrity cannot be preserved, users who should see the truth may become confused or the users who are supposed to be deceived, but without knowing it, may detect inconsistencies which reveal the existence of the deception.

4.1 Denying Existence

Denying the existence of a class of facts is relatively straightforward with any of the three techniques. Essentially the existence of the schema information which describes the sensitive facts must be hidden from the users who must not know such facts exist.

In all three approaches, the sensitive schema information is held inside Schema Objects, though these differ in format for each method. Only users whose clearance dominates the classification of a Schema can observe details of the schema. Without this information a user is unable to ascertain whether the information being hidden is of interest. So in most cases simply classifying the Schema Objects appropriately is sufficient to protect sensitive information.

However, in extreme cases it is necessary for the database to be constructed so that it does not reveal everything to some of its users, and yet they remain convinced that nothing is being withheld from them. A user who finds that they are unable to observe a Schema Object may be alerted to the fact that the database is not telling them the whole truth. Thus it is sometimes necessary to hide the existence of certain Schema Objects, not just their contents.

This, however, is the purpose of the Hierarchy. In the discussions about each approach, it was assumed that some control was being applied to prevent the creation of new Schema Objects from being used as a covert channel. Essentially this will be the use of either a special function which is trusted to not exploit the channel, or some extra level of Hierarchy which will hide the creation from users whose clearance does not dominate the clearance of the creator.

This extra level of Hierarchy can be used to ensure that users with insufficient clearances cannot detect the existence of schemas which they must not know exist. However, this is not a complete solution because these users will discover that there are things they are not allowed to see, although they will not necessarily know that they are schemas. Unfortunately, Bell-LaPadula style models do not lie about the existence of Objects (strictly, whether they are activated or not) and so the problem cannot be fixed within the simple modelling framework chosen here.

The requirement is for an Object whose existence cannot be detected by users of lower clearance. Thus, a user should receive the same error message if they attempt to access one of these hidden Objects as if they attempt to access a non-existent Object. The requirement can be implemented, though care must be taken to avoid covert timing channels which inadvertently reveal the two reasons for the same reply. However, in the absence of a suitable model this paper will leave the problem to future research and just indicate that existence needs to be hidden by drawing the Object's classification in outline format, eg. C.

To implement the example of section 2.2, two Schema Objects are required. Figures 4.1, 4.2 and 4.3 show this for each of the three techniques. One Schema lists those missiles for which the system records radiation information, the other records the actual radiation output for each of these missiles. The existence of both these Schemas is hidden inside a "directory" of Confidential schemas, thus an Unclassified user cannot detect that there is any information which they are not entitled to see. Confidential users will be able to discover that two schemas exist, but will be unable to determine what they describe.

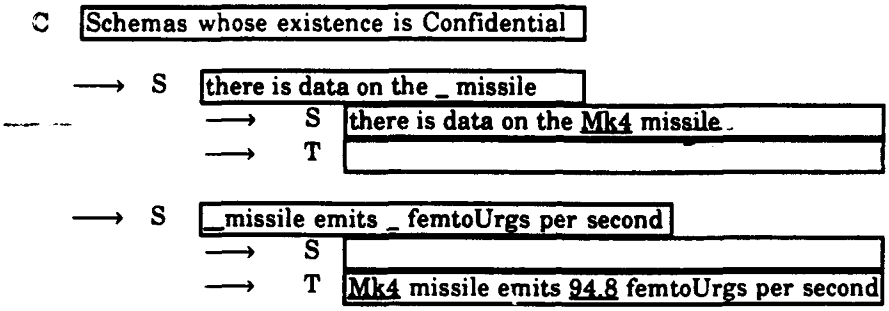
C
Schemas whose existence is Confidential
S
there is data on the _ missile
S
there is data on the Mk4 missile
T
S
_missile emits _ femtoUrgs per second
S
T
Mk4 missile emits 94.8 femtoUrgs per second

Figure 4.1: Denying the existence of information using polyinstantiation.

Note that if the Confidential "directory" was not hidden from lower users, they would be able to discover that an Object exists which they cannot observe. This is not usually a problem, but in this example the application wishes Unclassified users to believe there is nothing they cannot see.

Using polyinstantiation each Schema refers to a number of Collections, in this case two. Facts are stored in the Collection of the appropriate classification.

Using view based classifications, each possible fact is listed in the Schema. Associated with each possible fact is a Value Object which indicates whether the fact is true or not. The classification of the Value Object is what the classification of the fact would be if it were true.

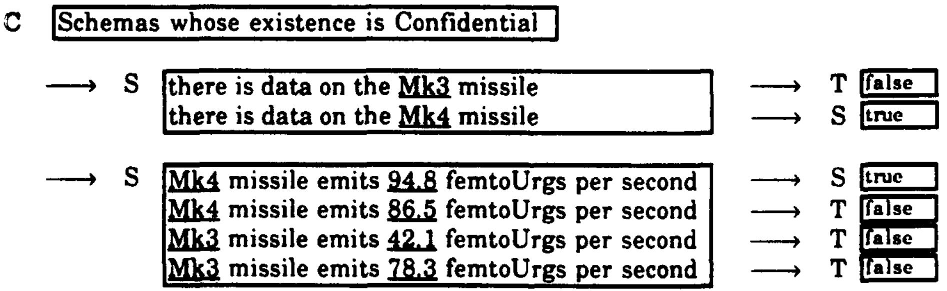
C
Schemas whose existence is Confidential
S
there is data on the Mk3 missile
there is data on the Mk4 missile
T
false
S
true
S
Mk4 missile emits 94.8 femtoUrgs per second
Mk4 missile emits 86.5 femtoUrgs per second
Mk3 missile emits 42.1 femtoUrgs per second
Mk3 missile emits 78.3 femtoUrgs per second
S
true
T
false
T
false
T
false

Figure 4.2: Denying the existence of information using views.

Using the Insert Low Approach, the facts are each stored in separate Fact Objects, which are referred to by the Schema Object. The fact's classification is that of the Fact Object which holds it.

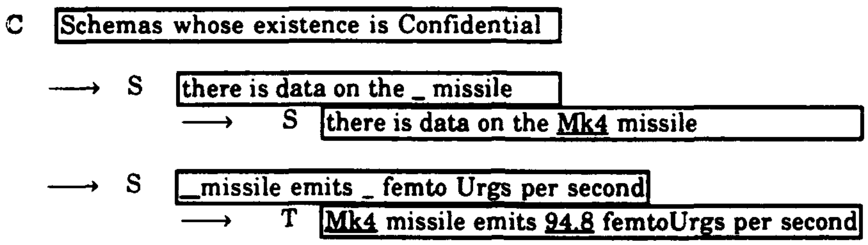
C
Schemas whose existence is Confidential
S
there is data on the _ missile
S
there is data on the Mk4 missile
S
_missile emits _ femto Urgs per second
T
Mk4 missile emits 94.8 femtoUrgs per second

Figure 4.3: Denying the existence of information using insert low.

So the ability of an application to deny the existence of information is independent of the chosen database security technique. The problem is solved by using the Hierarchy to hide the existence of schemas, though the notion of Hierarchy must be extended slightly to allow the existence of the hiding mechanism to be hidden.

4.2 Cover Stories

For a cover story to be effective, users who are being misled must not realise that the same kind of information is also held at a higher level, thus the techniques discussed above for denying the existence of information are necessary. Also, high users must be able to invent the cover story and enter this with a low classification. This necessitates the user logging in and out or the use of a multi-level workstation.

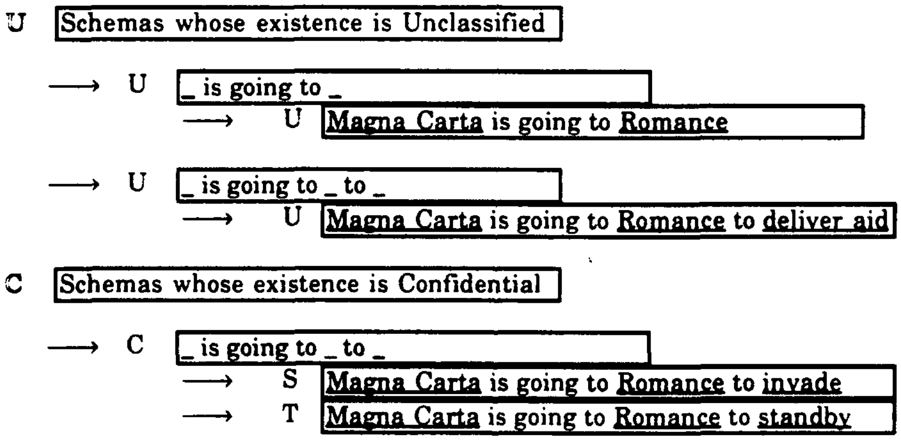
U
Schemas whose existence is Unclassified
U
_ is going to _
U
Magna Carta is going to Romance
U
_ is going to _ to _
U
Magna Carta is going to Romance to deliver aid
C
Schemas whose existence is Confidential
C
_ is going to _ to _
S
Magna Carta is going to Romance to invade
T
Magna Carta is going to Romance to standby

Figure 4.4: The problem with polyinstantiation for cover stories.

If these points are covered, any of the three techniques can be used to provide cover stories. This is possible because each can provide the ability to lie about the existence of information.

Using polyinstantiation, the main difficulty is not allowing cover stories, but preventing spurious ones from being created. Consider again the example shown in Figure 2.3, but suppose a Top Secret user now wishes to stand down the invasion force. The user is able to update the mission of the Magna Carta to *standby*, however, this change is not seen by any Secret user. This leads to disaster. Obviously the correct course of action for the Top Secret user is to log in at Secret and make the change, but the absence of any integrity check for duplicate missions means this is not enforced. Effectively a new cover story has been invented when one is not needed, as shown in Figure 4.4.

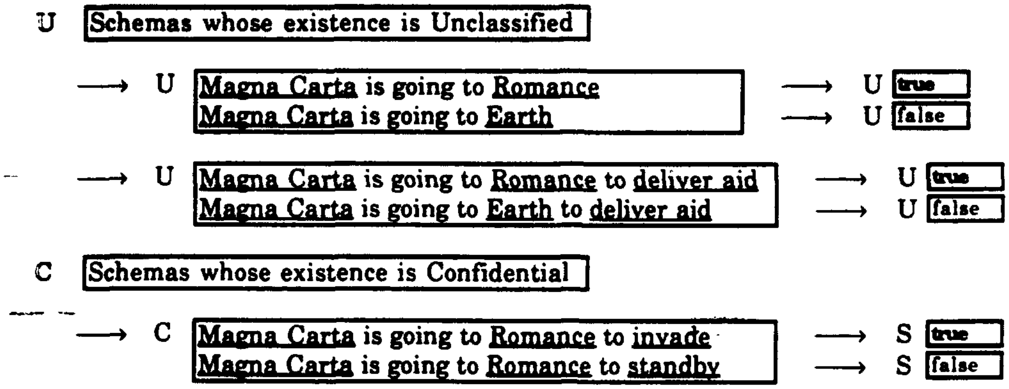
U Schemas whose existence is Unclassified
U Magna Carta is going to Romance
Magna Carta is going to Earth
U true
U false
U Magna Carta is going to Romance to deliver aid
Magna Carta is going to Earth to deliver aid
U true
U false
C Schemas whose existence is Confidential
C Magna Carta is going to Romance to invade
Magna Carta is going to Romance to standby
S true
S false

Figure 4.5: Cover stories using views.

View based classifications can be used to provide cover stories, as shown in Figure 4.5. Two mission schemas are created, one for the cover story and one for the real mission. The real mission is hidden from Unclassified users, while those with high clearances can see both but know the difference between the two. The view gives the classification of Magna Carta being put on standby as Secret, therefore a Top Secret user is prevented from updating it directly. By logging in at Secret the user could set the mission to standby, but integrity constraints would prevent this if the Magna Carta is already on an invasion mission, as in this example.

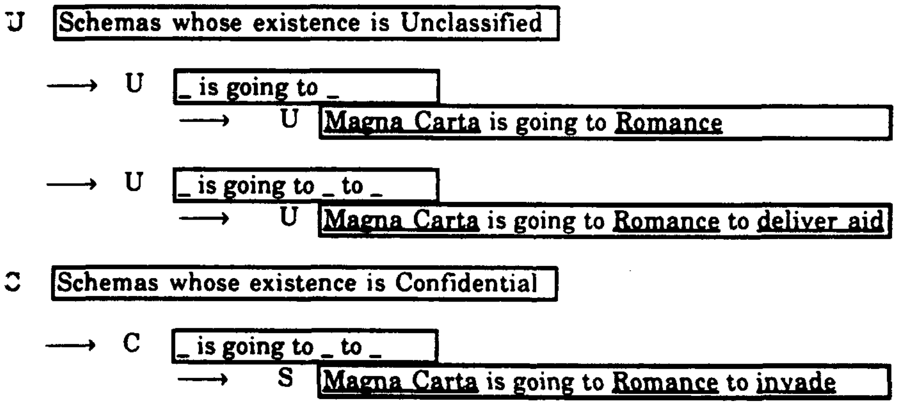
U Schemas whose existence is Unclassified
U _ is going to _
U Magna Carta is going to Romance
U _ is going to _ to _
U Magna Carta is going to Romance to deliver aid
C Schemas whose existence is Confidential
C _ is going to _ to _
S Magna Carta is going to Romance to invade

Figure 4.6: Cover stories using insert low.

The method for providing cover stories with the insert low approach is similar to that with view based classifications; two schemas are used, one for the cover story and one for reality. Figure 4.6 shows the example. The Top Secret user cannot change the mission because the Fact Object is Secret and this would constitute a downward flow. Once the user logs in at the Secret level the update and integrity checks present no problem.

So any of the three techniques provide the ability to invent cover stories. With Polyinstantiation cover stories can be invented "on the fly", while the other two techniques require design time decisions to build the ability to have cover stories into the schemas. Polyinstantiation therefore has the advantage that a cover story can be inserted without prior thought, but this is also its weakness because there is no way of preventing the creation of spurious cover stories.

4.3 Secrecy about Changes

If low information is to be altered by high users, and these changes are to be kept from low users, then the problem is similar to that for cover stories. However, the difference is that the low information is inserted by low users, which presents no problem. When the high users wish to update the information the original is left unaltered, and again this presents no real problem. Thus any of the three techniques for database security provide the ability to keep changes secret.

In fact the only difficulty is to ensure that the high user can refer to the correct information without confusion. Initially the low information is correct and this is what the high user will want to see. However, once this is superseded by high information, the high users will ordinarily wish to see the high information, though they may explicitly ask for the information as seen by low users.

So the solution is to use two schemas, one for the low version of the information and one for the high version when this is different. Low users have access to the low schema and may be prevented from detecting the high schema. High users have access to both schemas, but really wish they could see them as one. This can be achieved using a view mechanism, but the view definition will be complex and implementations usually insist that such views are read-only [ISO89].

Actually, all that is required is a special kind of project view, which is relatively simple to implement. This yields the high version of a fact if it exists, otherwise it yields the low version. When such a view is updated, the high version is updated, unless it does not exist in which case a high insert is performed. Unfortunately, such special purpose requirements are unlikely to be standardised, and so further research is necessary to investigate how updatable views can be defined in general.

5. Conclusions

In most secure databases classifying sensitive information is enough of a control, since the users will not be surprised to discover that there is information which is too highly classified for them to see. However, there are extreme cases where the existence of such information must be hidden. Further, the database may be required to deceive some users, by presenting them with a "cover story" rather than the truth. Similarly, the database may be required to hide changes made to some information from certain users.

Of the three techniques for achieving security in databases described in this paper, Polyinstantiation is the most widely known technique, but View Based Classifications and the Insert Low Approach are viable alternatives. Any of these three techniques could be used in a DBMS which supports databases that deceive their users. However, the requirement is not simply for a database that deceives, but for one where the deception is controllable.

This is where Polyinstantiation is weakest. Its inability to enforce even relatively simple integrity constraints means that cover stories can occur unintentionally. The resulting confusion could have serious consequences.

The two alternative approaches can both accommodate deception in databases, though in a controlled way. They are also, of course, able to enforce general integrity constraints even when deception is not required. As such, secure databases built using these techniques are likely to have superior data integrity characteristics to those that use Polyinstantiation.

However, both View Based Classifications and the Insert Low Approach do require more trusted code to implement the DBMS than with Polyinstantiation. As such it is more expensive to achieve a DBMS of a given assurance using these techniques. However, applications which use a Polyinstantiating DBMS may be more expensive to produce, because it is then up to the application to enforce integrity.

In summary, Polyinstantiation is not the only technique for achieving security in databases, even when the database is required to deceive some users, and the alternatives merit further attention. Future research must devise database design methods for each database security technique to allow fair comparisons to be made as to their effectiveness.

References

[Bell74]
"Secure Computer Systems: A Refinement of the Mathematical Model"
D E Bell & L J LaPadula, MTR-2547, Vol 3, Mitre Corp., April 1974

[Burns 90]
"Referential Secrecy"
R K Burns
Procs. IEEE Symp on Security and Privacy, Oakland, CA, May 1990, pp133..142

[Cummings 87]
"Compartmented Mode Workstation: Results Through Prototyping"
P Cummings, D Fullam, M Goldstein, M Gosselin, J Picciotto, J Woodward & J Wynn
Procs. IEEE Symp on Security and Privacy, Oakland, CA, April 1987, pp2..22

[Denning 87]
"A Multilevel Relational Data Model"
D E Denning, T F Lunt, R R Schell, M Heckman & W R Shockley
Procs. IEEE Symp on Security and Privacy, Oakland, CA, April 1987, pp220..234

[Denning 88]
"The Sea View Security Model"
D E Denning, T F Lunt, R R Schell, W R Shockley & M Heckman
Procs. IEEE Symp on Security and Privacy, Oakland, CA, April 1988, pp218..233

[Garvey 88]
"ASD-Views"
C Garvey & A Wu
Procs. IEEE Symp on Security and Privacy, Oakland, CA, April 1988, pp85..95

[Haigh 90]
"The LDV Secure Relational DBMS Model"
J T Haigh, R C O'Brien & D J Thomsen
Procs. IFIP WG11.3 Database Security Workshop, Halifax, England, September 1990

[ISO89]
"Information Processing Systems Database Language SQL with Integrity Enhancement"
ISO/IEC 9075:1989(E)

[Jajodia 90]
"Polyinstantiation Integrity in Multilevel Relations"
S Jajodia & R Sandhu
Procs. IEEE Symp on Security and Privacy, Oakland, CA, May 1990, pp104..115

[Keefe 90]
"Multiversion Concurrency Control for Multilevel Secure Database Systems"
T F Keefe & W T Tsai
Procs. IEEE Symp on Security and Privacy, Oakland, CA, May 1990, pp369..383

[Knode 88]
"Making Databases Secure with TRUDATA Technology"
R B Knode & R Hunt
Procs. 4th Aerospace Comp Sec Applications Conf, Orlando, Florida, Dec 1988, pp 82..90

[Lunt 89]
"Aggregation and Inference: Facts and Fallacies"
T F Lunt
Procs. IEEE Symp on Security and Privacy, Oakland, CA, May 1989, pp102..109

[Martin 90]
"Enterprise Modelling and Security Policies"
M Martin & J Dobson
Procs. IFIP WG11.3 Database Security Workshop, Halifax, England, September 1990

[Morgenstern 88]
"Controlling Logical Inference in Multilevel Database Systems"
M Morgenstern
Procs. IEEE Symp on Security and Privacy, Oakland, CA, April 1988, pp245..255

[Sowerbutts 90]
"Database Architectonics and Inferential Security",
B J Sowerbutts & S Cordingley
Procs. IFIP WG11.3 Workshop of Database Security, Halifax, England, Sept 1990

[Terry 89]
"A 'New' Security Policy Model"
P Terry & S Wiseman
Procs. IEEE Symp on Security and Privacy, Oakland, CA, May 89, pp215..228

[Wilson 88]
"Views as the Security Objects in a Multilevel Secure Relational DBMS"
J Wilson
Procs. IEEE Symp on Security and Privacy, Oakland, CA, April 1988, pp70..84

[Wiseman90a]
"Control of Confidentiality in Databases"
S R Wiseman
Computers & Security Journal, Vol 9, Num 6, October 1990, pp529..537

[Wiseman90b]
"The Control of Integrity in Databases"
S R Wiseman
Procs. IFIP WG11.3 Database Security Workshop, Halifax, England, Sept 1990

[Woodward 87]
"Exploiting the Dual Nature of Sensitivity Labels"
J P L Woodward
Procs. IEEE Symp on Security and Privacy, Oakland, CA, April 1987, pp23..30

REPORT DOCUMENTATION PAGE

DRIC Reference Number (if known) ....

Overall security classification of sheet .........UNCLASSIFIED.........

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the field concerned must be marked to indicate the classification, eg (R), (C) or (S).

Originators Reference/Report No. MEMO 4503 (ISSUE 2)	Month APRIL	Year 1992

Originators Name and Location
DRA, ST ANDREWS ROAD
MALVERN, WORCS WR14 3PS

Monitoring Agency Name and Location

Title

LIES, DAMNED LIES AND DATABASES
(ISSUE 2)

Report Security Classification UNCLASSIFIED	Title Classification (U, R, C or S) U

Foreign Language Title (in the case of translations)

Conference Details

Agency Reference	Contract Number and Period
Project Number	Other References
Authors WISEMAN, S R	Pagination and Ref 19

Abstract

A database is usually expected to give correct and complete answers to queries. However, some applications take confidentiality to an extreme and require the database to deceive some users by supplying incorrect answers. This paper examines these requirements and studies the effectiveness of three database security techniques in this area.

Abstract Classification (U, R, C or S)
U

Descriptors

Distribution Statement (Enter any limitations on the distribution of the document)
UNLIMITED